The Benefits and Challenges of Executable Acceptance Testing

Shelly S. Park
University of Calgary
2500 University Dr. NW
Calgary, AB, Canada, T2N 1N4
+1 (403) 220-3535
parksh@cpsc.ucalgary.ca

Frank Maurer
University of Calgary
2500 University Dr. NW
Calgary, AB, Canada, T2N 1N4
+1 (403) 220-3531
maurer@cpsc.ucalgary.ca

ABSTRACT
In this paper, we argue that executable acceptance test driven development (EATDD) allows tighter integration between the software requirements and the implementation. We argue that EATDD improves communication between all project stakeholders. We give an overview of why previous approaches to requirements specifications are less than impressive and how executable acceptance tests help fix problems. In addition, we argue for multi-modal executable acceptance tests and how it can help improve the requirements specification. We provide some of the immediate research questions that need to be addressed in order to push forward more wide-spread use of executable acceptance test driven development.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications, Testing and Debugging

Keywords
Executable Acceptance Test Driven Development, Automated Software Testing, Agile Software Engineering, Executable Requirements

1. INTRODUCTION
Executable Acceptance Test Driven Development (EATDD) emphasizes writing software requirements in form of executable acceptance tests. One of the biggest contributions of Agile methodologies is the concept of test-driven development. In test-driven development, the tests are written before writing the actual code. The tests can be used to evaluate the development progress by measuring the number of passing or failing tests and to perform continuous regression testing, which can help maintain high software quality by notifying the developers of software defects as soon as the code is changed. TDD was introduced for the unit test level and helps with designing the software and its detailed APIs. In an EATDD environment, TDD is extended from the software design to the requirements/specification level: requirements are written in form of executable acceptance tests or executable specifications. The term "executable specification" here denotes the combination of a (natural language) test definition plus the fixture code needed to execute the test. This is slightly different than the meaning in algebraic or logical specification where a general execution engine directly executes the test definition. Executable acceptance tests are developed by the customer in collaboration with Business Analysts (BA)s and/or Quality Assurance Engineers (QA)s. The tests define the criteria for an acceptable system from the customer/business perspective and can be used as a contract between the development team and the business stakeholders.

EATDD is a process that can help communicate requirements. In an EATDD environment, acceptance tests play an important role throughout the software development cycle, because everyone in the software development team is involved in writing, maintaining, testing and developing the tests at some point in their work. The automated executable tests give confidence to the customer that the requirements are understood correctly. The execution of these tests can be used as a real-time indicator of the development progress and a live documentation about the software. By making the tests executable, the developers and business stakeholders are forced to be concrete and unambiguous about requirements. This avoids some well-known problems with text-based or diagram-based requirements specifications.

In our opinion, test-driven development and continuous integration can help the team be more productive in the long run. The test driven development should be practiced not just for the developers, but for everyone involved in the team, which is where EATDD comes in. EATDD allows people from different functional roles to engage in test-driven development from all software development lifecycle perspective including requirements analysis, testing, coding as well as overall project progress tracking.

There are several tools that support EATDD. For Agile practitioners, Fit[1] or Fit-based tools such as FitNesse [2] are popular executable acceptance testing tools. On the user interface side, there are tools such as WATIR [3] or Selenium [4] that are designed to automate human testing by automating the testing of the user interface layer. These tools serve the purpose, but we need something better in order to drive EATDD to the next step

APSO'08, May 10, 2008, Leipzig, Germany.

of adoption. There is a lack of better software requirements and testing tools, which is a difficulty all practitioners face no matter which method is used. Development teams in more traditional environments can also appreciate the difficulty in defining the right requirements and maintaining the requirements document for future changes.

This position paper argues that executable acceptance testing is a good way to specify software requirements and that it improves communication between all project stakeholders. Section 2 discusses the background information about requirements analysis and requirements specification process. Section 3 discusses the current state of executable acceptance testing and Section 4 discusses the need for multi-modal executable acceptance testing. Section 5 concludes the paper by arguing for more research and tools development that can help improve EATDD.

2. BACKGROUND

Requirements analysis is traditionally communicated to development teams through documents. The document may contain use cases or just detailed descriptions. There is IEEE standard for requirement specifications [21], but it is uncertain how many people actually follow the standard explicitly. Requirements analysts do not have as effective tools as developers in defining and maintaining requirements specifications [5]. As the system becomes more complex, analysts spend more time on requirements specifications and requirements maintenance and a lack of good tools is not helping with the problem [6]. It is estimated that requirements errors contribute about 25 to 70 percent of the total software errors [7]. Two thirds of requirements errors are undetected until after the delivery and requirements errors are these errors are very costly to fix [8]. As Neumann argues, requirements errors are hard to detect because of interactions between the requirements; sometimes the requirements can conflict with each other, undoing and complicating what other requirements had done [9]. Often these requirements conflicts are hard to detect until there is actual testable software and these strange software behaviors are difficult to discover until the software is actually used by people.

Requirements also change over the lifetime of the software. Some of the changes are inevitable due to changing environmental factors such as emergence of new technology or need for more marketable features in the software. As Crnkovic et al. mention, "inadequate use of requirements and inadequate guidance by requirements" can lead to loss of control over the requirements changes and the actual state of the implementation [10]. It is impossible to validate and verify the software implementation against the requirements if the requirements specifications are no longer in synch with the actual implementation.

Requirements analysis is a very difficult problem to solve because of the nature of the requirements elicitation process. For example, Stone and Sawyer argue that requirements specification is about converting tacit knowledge into explicit knowledge. Being able to identify which tacit knowledge to code into software requirements or even realizing how different tacit-knowledge can impact other requirements can be a tricky problem to solve [11]. Having wrong assumptions about tacit knowledge is hard to catch and these problems generally don't surface until after the software is implemented and used by people. As a rule, the problems that are noticed by the users are the costliest errors to fix. Finding ways to deal with wrong assumptions as early as possible in the software development cycle is the best way to minimize the number of errors and costly changes at the end.

Requirement documents, as traditionally exercised by waterfall approach, are deficient in many ways. About 13% of the requirements contain information that is irrelevant to the actual problems [12] and 29% of the requirements are missing in the requirements document [13]. In addition, some of the notorious problems include over-specification, wishful thinking, ambiguities and lengthy documents [14, 15].

The dilemma keep being presented in all of these requirements-related problems is that requirements errors are not obvious to detect until the software is available and new versions have to be checked against all old requirements – which makes regression testing time consuming and costly. Second, the requirements specification might become outdated during the lifetime of the software. Besides the fact that a requirements document is less useful if it is no longer up-to-date, spending a lot of time inspecting the requirements document for completeness or trying to find conflicting requirements from the document also doesn't seem to work without having software that users can try out. In addition, in a waterfall approach, requirements errors that are not detected during the requirements phase will not be detected until the testing phase and will become very expensive to fix.

A solution is to repeat the development cycle in small incremental iterations, as recommended by Agile methods. The concept of incremental iterations in software development existed for many years such as Boehm's spiral model [16]. However, the difference is that Agile methods, such as eXtreme programming, emphasize developing automated tests before the development. There is no point in spending too much time in requirements specifications phase if large number of requirements errors are not detected until the testable software is available. By writing the tests before the code, we can verify that the code is behaving according to the assumption laid out in the tests. In test driven development, the concept of testing is no longer just reserved for QAs. Developers must write the unit tests before writing the code. BAs no longer write documents, but they write executable acceptance tests along with the customer and QAs. Infecting everyone to think about testing first allows better software quality and opens up opportunities for better communication.

However, the challenge of introducing EATDD to a team with no prior test-driven development experience is the concept that test automation is not test-specification automation [22]. It takes time and effort to write and maintain these tests. Without a well disciplined team, the tests could also get out of sync with the code if the team doesn't maintain the test code properly. It is also possible that the team is not trained properly enough to write executable specifications or test automation scripts. This is the same problem that test-driven development faces in general.

In an EATDD environment, requirements specifications are no longer just an artifact that gets forgotten during the planning phase and never gets read or updated. The requirements specification in form of tests can organize, communicate and validate the implementation in real-time and can be used as a live documentation about the state of the development progress.

3. EMPIRICAL STUDIES ON EATDD

There are many benefits to EATDD, but executable acceptance testing is not practiced as much as other Agile techniques, mainly due to lack of good tools and its relative newness. There is big expectation about the possibilities of EATDD; people are often excited when they hear about the concept. Some of the empirical studies show that the current tools fulfill its basic purpose, but there is still a need for better tools. It seems that, although we don't have strong evidence to back up our claim, a lack of better and more efficient tools are making people cautious about practicing EATDD.

By far, the most popular tool is Fit [1] or Fit-based tools such as FitNesse [2]. A study done by Read et al. shows that over two thirds of the computer science students who tried Fit responded positively on using Fit for future projects [17]. Similarly, about half of the business students responded positively on the use of Fit [18]. An interesting finding from [18] is that those who complained most about Fit actually wrote one of the best executable acceptance tests. A common theme from these complaints is that the participants were surprised at the amount of work that is involved in specifying these tests. Some of the tests were so simple that the participants actually felt just a verbal communication would have been easier. [18] also reports on interviews with industrial users. The comments from the industrial users about their experience with executable acceptance tests show that the communication between different stakeholders has improved because disagreements became visible from the beginning. The tests allow more rapid feedback about the development progress and requirements-related defects were dramatically reduced. These studies were done with only a small group of volunteers and we are still cautious to generalize the result to more wide-spread situations without gathering more evidence. However, the result shows some positive feedback about how Fit can be used effectively.

In October 2007, Agile Alliance held a workshop on Functional Testing Tools [19]. The participants in the workshop have identified nearly a hundred items that still need more research and development. Some of these problems include how to define tests, how to create a common test format, organize/maintain large complex tests, overcome the limitations of natural language, make the tools and test specifications easier to understand for customers, expressing tests for GUIs, getting BAs and customers to write the tests (and actually make them enjoy doing it), writing more robust tests, create a super power IDE for BAs, maintain more sustainable communication between all stakeholders and many more. The problems identified during the workshop are quite open-ended research questions and very complex issues because of their open-ended nature. They involve not only technological improvements, but also sociological improvements that challenge the current way people think about software engineering, especially requirements specifications.

4. MULTI-MODAL TESTING

Executable acceptance tests have many users and audiences. There are customers, BAs, QAs, developers, a project manager and others who are involved in the project with varying degrees of responsibilities and interest in the project. They have different background skills and different domain knowledge. In case with internal software development projects, different departments within the company may have competing agendas and have different reasons for wanting the development project to be successful. Facilitating communication between these people to gather software requirements, design decisions, project progress can be a very difficult task especially in a larger team. Rather than perceiving requirements document as a separate artifact, requirements specification should be the driver of all aspects of the software development including progress tracking, task assignment, quality assurance and information repository.

Different people have preferences for different kinds of notations and perhaps different kinds of notations are better suited for explaining different kinds of concepts. A requirements specification tool must be flexible enough to annotate information in diverse formats. For example, diagrams may explain graphical information much better than words. The real catch is how to make these different notations to be testable. The next generation of acceptance testing tool needs to accommodate a wide range of users and fulfill their communication needs. The next generation of acceptance testing tools needs to be multi-modal: tests need to be expressible in multiple formats in order to represent the requirements specification to different stakeholders in a format that is easily understood by them.

Developers use executable acceptance tests to get rapid feedback about the correctness of the implementation. Developers can also convey the design decisions to the QAs through executable acceptance tests such as diagrams of software architecture.

QAs and BAs can use executable acceptance testing tools as a place holder for all information that they gather such as links to documents, websites or data files. The tests should be able to hold any non-functional requirements such as platforms requirements or runtime libraries. By being able to associate these artifacts with the tests, the testing tool becomes one-stop repository for information that helps communicate everyone's needs better to all stakeholders.

Project managers can obtain real-time progress information and allocate the resources appropriately. When the customer approaches with new requirements, the impact of the changes can be quickly assessed by changing the executable acceptance tests and showing the magnitude of changes through failing tests. Showing more concrete evidence can help with feature negotiations as everyone can clearly see the number tests that are failing in order to implement the new feature.

There is a give and take relationship between flexibility in notation and the ability to create automated tests. For example, the table format that Fit uses for its test specification is easy to automate. However, some workflow information is better to represent in diagrams such as Swimlanes, UML activity charts, wireframes, or pictures. Is there an easy way to automate these non-textual requirements? What if we want to switch our views between different representations? The first solution is to provide a place to put all these artifacts into a test holder so these artifacts are accessible to everyone. However, if we cannot turn these artifacts into testable components, they are all in the risks of becoming out-dated artifacts that everyone will forget. The ability to represent and switch between different notations for a single test is what we call multi-modal executable acceptance test specifications.

A second issue is the organization of large numbers of tests. When the number of tests increases, a simple tree structure of folders and tests may not be sufficient in getting the overall project vision across. It is hard to detect duplicates, missing tests, and the workflow of the entire system. An efficient search algorithm is required as well as a good visual representations of the tests. In addition, these tests may become too large to maintain easily. In other words, we need good test management and refactoring tools, which is what tools such as FitClipse [20] are trying to achieve. The future of tool development for better use of EATDD is filled with possibilities. We have addressed some of the immediate questions that need answering in this section. There is no doubt in our mind that good tools and better informed practitioners will help spread the practice EATDD and Agile methodology.

5. CONCLUSION

Executable acceptance test driven development provides promising possibilities on requirements specifications. It finally allows software development to be driven by the requirements, rather than losing the requirements perspective as the development progresses. It directly links requirements and QA. The tight integration between the executable acceptance tests and the code allows better progress tracking, better communication between stakeholders and better software quality. We need to build better tools to help push forward EATDD and perform more empirical studies confirming or rejecting existing findings. Our immediate goal is to find out how to improve the tools to become multi-modal. There are many possible solutions to tool improvements, which will help benefit our knowledge about some of the possible ways requirements can be specified and possibly even make our BAs and customers to enjoy writing executable acceptance tests.

6. REFERENCES

[1] Fit, http://fit.c2.com/

[2] Fitnesse, http://fitnesse.org/

[3] Watir, http://wtr.rubyforge.org/watir_user_guide.html

[4] Selenium, http://www.openqa.org/selenium/

[5] Lempp, P., Rudolf, L.,1993, What Productivity Increases to Expect from a CASE Environment: Results of a User Survey, *Computer Aided Software Engineering (CASE)*, Ed. Chikofsky, IEEE, 1993, pp. 147- 153

[6] Graf, D.K., Misic, M.M., 1994, The Changing Roles of the Systems Analyst, Information Resources Management Journal, 7 (2), Spring, 1994, 15-23.

[7] Jones, C., 1995, Software Challenges, *Computer*, Vol. 28, No. 10, October 1995.

[8] Boehm, B., 1981, Software engineering economics, Prentice-Hall, Englewood Cliffs, N.J.

[9] Neumann, P.G., 1995, Computer Related Risks, Addison-Wesley

[10] Crnkovic, I.; Funk, P.; Larsson, M., 1999, Processing requirements by software configuration management, Proc. 25th EUROMICRO, Volume 2, 8-10 Sept. 1999, 260 - 265 vol.2

[11] Stone, A.; Sawyer, P., 2006, Identifying tacit knowledge-based requirements., Proc. IEEE Software, Volume 153, Issue 6, Dec. 2006, pp. 211 - 218

[12] Hooks, I., Farry,K., 2001, Customer-Centered Products:Creating Successful Products Through Smart Requirements Management. American Management Association, New York, NY

[13] Fowler, M., Specification By Example, http://www.martinfowler.com/bliki/SpecificationByExample.html

[14] Davis, A., 1994, Software Requirements Revision Objects, Functions & States. Prentice Hall PTR, Englewood Cliffs, NJ

[15] Meyer, B., 1985, On Formalism in Specifications, IEEE Software, Vol. 2, No. 1, pp. 6-26

[16] Boehm, B., 1986, A spiral model of software development and enhancement, ACM SIGSOFT Software Engineering Notes, Vol. 11, Iss.4

[17] Read, K., Melnik, G., Maurer,F., 2005, Students experiences with executable acceptance testing, Proceedings of the Agile Development Conference 2005, Denver, Colorado, USA

[18] Melnik, G., 2007, Empirical Analyses of Executable Acceptance Test Driven Development, PhD Thesis, University of Calgary

[19] Agile Alliance Functional Testing Tools Visioning Workshop, Oct 2007, http://www.agilealliance.org/show/1938

[20] FitClipse, http://sourceforge.net/projects/fitclipse/

[21] IEEE Recommended practice for software requirements specifications, http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=15571

[22] Martin, R., Melnik, G., 2008, Tests and Requirements, Requirements and Tests: A Mobius Strip, IEEE Software, 25(1), pp. 54-59